ActiveRecord in Rails

Database Connectivity Rails Way

Sumanth Krishna. A
Blog: TechSavvy
http://sumanthtechsavvy.blogspot.com

Ruby on Rails

We always talk about programming languages and we take pride in propagating how the language follows/supports the object oriented paradigm. This really makes sense, since we can connect them to real time objects. But, what about the databases? All the web applications do require some databases to store the content, user information and so on. We alway depend on database for any kind of information, but

Are our databases are ready to listen to objects?
How the languages communicate to the database (nothing but content)?

Well this is where the Active Record pitches in: Active record is an approach that is used to access the data.
How it does?

Programming Language Database

class	=====>	*table*
object	=====>	*single row in table*
properties/methods	=====>	*columns in table*

"A database table or view is wrapped into a class, thus an object instance is tied to a single row in the table. After creation of an object, a new row is added to the table upon save. Any object loaded gets its information from the database; when an object is updated, the corresponding row in the table is also updated. The wrapper class implements accessor methods or properties for each column in the table or view."

Now that is with Active record. Rails do implement the same pattern and with power of ruby, it does add some extra features/conventions to make developers life simpler.

Well in following posts we will see,

1. How active record is implemented in rails
2. What are the conventions?
3. How to configure database to rails applications?
4. Do we need to write xml files to configure each table?

Sumanth Krishna. A

Ruby on Rails

One of the main features that rails do support on is Convention over Configuration. Well repeat again and again Convention over Configuration. Let it get into the minds!
Well it is spread across the rails framework with which will be developing an application. I will be reminding whenever it's put in use.
In previous post, I did stress on how simple it would be in configuring the databases and tables with your application. In Java based applications, this would be done using lot of xml files, which is of course, time taking, lot of repetitive work and error prone... (at least that's what my friend's used to say!).

Well in rails this is not the case a simple convention that you follow would ensure of configuring the tables of databases.

> "You name the tables in database as plurals and create classes with singular form of the table names."

Configuring the Databases:
The choice of which database you use the configuration and respective adapter would depend on. When we created an application, there is a set of predefined folders created over a simple command! (I will touch upon this later). "config" is one such folder and under this we will now concentrate/repair on "database.yml".
Rails would automatically create 3 different environments:

1. Development
2. Test
3. Production

Based on the environment we work on we need to edit the data accordingly. (More on environments will be covered sooner!)
This file contains the information relate to the adapter, database, user credentials, host, port...

This does make sense in a way, the plural form for tables will fit as it contains huge data and the respective class which we would be placed under app/models, with the extension ".rb" is named in singular form.

```
development:
    adapter: mysql
    database: dummy
    username: root
    password: admin
    host: localhost
```

What it means?
Our application is connected to MySQL database by name dummy, user being and root and with an empty password. The host environment is "localhost".

Sumanth Krishna. A

Ruby on Rails

To Be Continued...

Get ready for more surprises...

1. Configuring tables
2. Relationships between tables
3. ActiveRecord implementation in rails
4. Conventions that rails uses.

Tag Cloud:

ActiveRecord xml Rails configuration relationships Ruby on Rails one-to-one many-to-many tables one-to-many class **keywords** convention Ruby **has_one** belongs_to **host** has_many **adapter** **database** mysql pgsql **localhost** has_many :through habtm production environment

Sumanth Krishna. A

Ruby on Rails

In the previous post, I just shown how do you tell your application the whereabouts of the database(s) that's in use. Now we are left out with task to connect each tables. Let me remind you again, this would need configuring/customizing lots of xml files in java based applications.
You may ask:
Come on, How do you achieve it without the xml files?

Well the magic here is usage of, pluralisation of tables and respective singular form for it's class names.

Can you show me how to do it?
hmmm... of course.

In last post we modified database.yml of our "dummy" project.

```
adapter: mysql
database: dummy
username: root
password:
host: localhost
```

Just recollect, this shows our "dummy" project is referencing to database by name "dummy". I am creating the database related tables, objects (rows), columns (properties) using a tool called phpMyAdmin.

Assuming that this "dummy" database contains a table with user information. So our convention would suggest to have table name in plural form. Let's call it as "users". Now we need to have a class created with singular form of the "users" -> "user", and respective logic relating to the table would go into/under this class.

```
class User < ActiveRecord::Base

end
```

> ***Let us diagnose a bit the above code. Recollect the convention we talked above, and the ORM concept. This means that the applications having class called "User" which would connect/contact to database table by name "users" (the database info is set up in database.yml).***
>
> ***We will slowly getting into the code and we keep developing or adding more functionalities to our "dummy" project.***
>
> ***Do I need to use only mysql as the database?***
> ***Not exactly and need not be. The following table would give the info related to configure each database and the respective parameters:***

Sumanth Krishna. A

Ruby on Rails

:adapter ⇒	db2	mysql	oci	postgresql	sqlite	sqlserver
:database ⇒	*required*	*required*		*required*		*required*
:host ⇒		localhost	*required*	localhost		localhost
:username ⇒	✓	root	*required*	✓		sa
:password ⇒	✓	✓	*required*	✓		✓
:port ⇒		3306		5432		
:sslcert ⇒		✓				
:sslcapath ⇒		✓				
:sslcipher ⇒		✓				
:socket ⇒		/tmp/mysql.sock				
:sslkey ⇒		✓				
:schema_order ⇒				✓		
:dbfile ⇒					*required*	

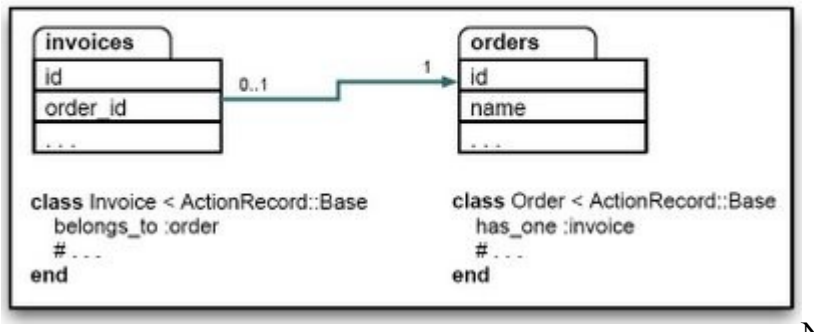
Well more action and fun is on the way :)

Keywords:

- has_one,
- belongs_to,
- has_many,
- has_and_belongs_to_many
- polyphormic
- one-to-one
- one-to-many
- many-to-many

It's known that any web application will contain bunch of tables and they are dependent on each other. With normalization as the key, we avoid redundancy and have more relationships with tables using keys. In database terms, the following are the relations allowed/known between tables -> one-to-one, one-to-many, many-to-many. Let us digg more, how ActiveRecord handles these.
ActiveRecord supports the above mentioned relations and in fact, comes with set of key words which are easy to use, understand and pretty clean.
Let me take few snapshots from David's book:

One-to-One Associations



Note:

1. invoices is the table name
2. orders is the table name

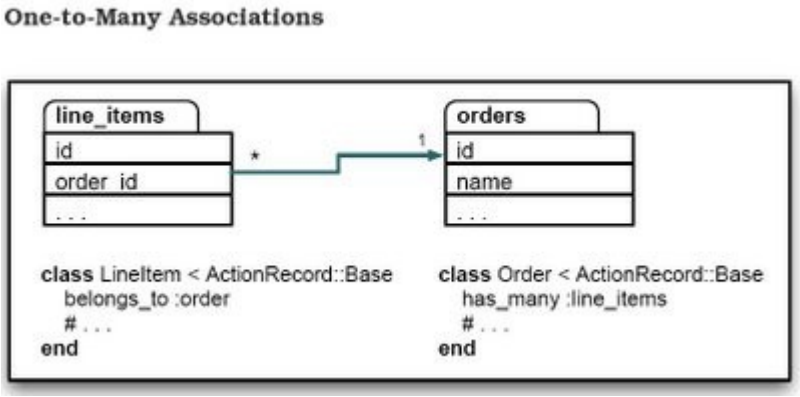
Sumanth Krishna. A

In the above snaphot, the motive is to declare the relationship between the orders and invoices tables. It's clear that orders will have invoices, i.e., invoices belong to orders. And watch out we will be using same words. Now to have the relationship declared, we will go to their models (Remember: the application access the tables through model so it makes sense to declare relationships also in models).

Now I am into order model, orders having invoices. so the key word that we use here is has_one and then followed by the model name (Note: It is not table name, this is not plural).
has_one :invoice

Next will go to invoice model, since invoices belongs to orders we use the key word belongs_to and then followed by the model name.
belongs_to :order

Well there is not xml written for models, no xml written for relationships. But simple convention and defined keywords done the job for us.
And as you might have guessed the has_many comes into use when the orders table is having relationship with one more table (say, line_items). The below snapshot will help you to understand the syntax and it's usage.

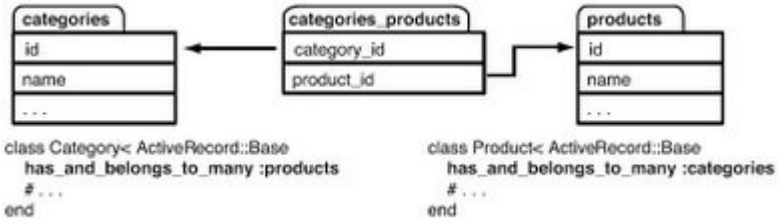


Will look into other keywords and other relationship in the following posts...

has_and_belongs_to_many, simply "habtm" of rails would address/handle the many to many relationships of the tables.

Let us take this scenario where we have two tables, "products" belong to "categories" and "categories" belongs to "products" in more than one ways. That is each product belongs to many categories and each category contains many products. In each of the model, category and product the relation would be shown using habtm (has_and_belongs_to_many).

Ruby on Rails



Closely observing the above example, we can see that the products table and categories table contains habtm relation. There is an intermediate table "categories_products", which would carry the primary key of each table.

At the database level this is achieved using an intermediate join table. The convention followed here is the name of the join table is the concatenation of the two target table names but in alphabetical order. In our example, we joined the table categories to the table products, so Active Record will look for a join table named categories_products.

Note:

1. The join table carry the foreign keys of each table.
2. There will be no model for this intermediate table (categories_products)

Well that precisely explains how we implement many to many relationships in Rails. That is not all there is one more way of expressing the relationship between tables, "has_through" which we will touch base later.

We had seen how a has_and_belongs_to_many works with a join table. But apart from carrying the foreign keys the join table has nothing much to do over there. So to have more features added and retaining the goal of having many-to-many relationships we will discuss now another rails offering "has_many, :through".

```
class Book < ActiveRecord::Base
  has_many :contributions, :dependent => true
  has_many :contributors, :through => :contributions
end
class Contributor < ActiveRecord::Base
  has_many :contributions, :dependent => true
  has_many :books, :through => :contributions
end
class Contribution < ActiveRecord::Base
  belongs_to :book
  belongs_to :contributor
end
```

Join Table

The above snippet shows a simple example from Josh Susser's blog.
The scenario shows that there are two tables "books" and "contributors" and the a join table

Sumanth Krishna. A

"contributions". This is entirely different from habtm, where we had the join table with combination of tables.
And syntactically, we say to each of tables (books, contributors) that they are related to each other through "contributions".
So the join table would have a simple belongs_to and the individual tables will be related with has_many:'table_name' and keyword :through=>'name_of_join_table'

That is with the ActiveRecord relationships. Just to recap
has_and_belongs_to_many
has_many, has_one, belongs_to

Ruby on Rails

This document is a part of the Ruby on Rails course structure that I had started @ TechSavvy my technical blog. This document is made keeping the freshers to the industry as well as starters of RoR and its implementations.

Http://sumanthtechsavvy.blogspot.com

Download other reseouces:

http://tosumanthkrishna.googlepages.com/RubyonRailscoursestructure.pdf
http://tosumanthkrishna.googlepages.com/RubyonRailscourse1-WhatisRuby-.pdf
http://tosumanthkrishna.googlepages.com/RubyonRailscourse1.2.pdf
http://tosumanthkrishna.googlepages.com/RubyonRailsInstallingRuby.pdf
http://tosumanthkrishna.googlepages.com/RubyonRailssimpleexamplesstrings.doc

View Resources online:

http://www.scribd.com/doc/733856/Ruby-on-Rails-course-structure
http://www.scribd.com/doc/830473/Ruby-on-Rails-Lesson1-What-is-Ruby
http://www.scribd.com/doc/883390/Ruby-on-Rails-Learning-ruby-through-examples
http://www.scribd.com/doc/887331/Ruby-on-Rails-Installing-Ruby
http://www.scribd.com/doc/918737/Ruby-on-Rails-simple-examples-strings

Sumanth Krishna. A